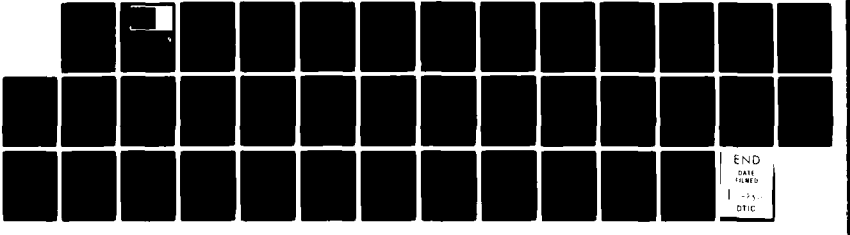
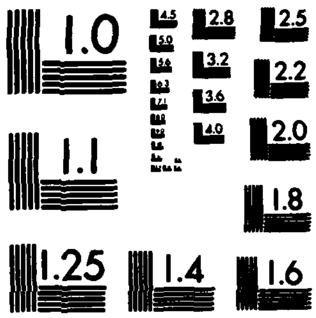
AD-A120 959 DIFFERENTIATION IN PASCAL-SC: TYPE GRADIENT(U) 1/1
WISCONSIN UNIV-MADISON MATHEMATICS RESEARCH CENTER
L B RALL JUL 82 MRC-TSR-2400 DAAG29-80-C-0041

UNCLASSIFIED F/G 12/1 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

(2)

AD A120959

MRC Technical Summary Report #2400

DIFFERENTIATION IN PASCAL-SC: TYPE GRADIENT

L. B. Rall

Mathematics Research Center
University of Wisconsin—Madison
610 Walnut Street
Madison, Wisconsin 53706

July 1982

(Received June 21, 1982)

Approved for public release
Distribution unlimited

Sponsored by

U. S. Army Research Office
P. O. Box 12211
Research Triangle Park
North Carolina 27709

UNIVERSITY OF WISCONSIN-MADISON
MATHEMATICS RESEARCH CENTER

DIFFERENTIATION IN PASCAL-SC: TYPE GRADIENT

L. B. RALL

Technical Summary Report #2400
July 1982

ABSTRACT

In scientific computation, there is often need for the derivatives as well as the values of functions defined by computer programs. Here, it is shown how automatic differentiation can be carried out in a modern computer language which permits user-defined operators and data types. The specific language used is PASCAL-SC, and differentiation is implemented for variables of type GRADIENT, which consists of the value of a function of n real variables and its gradient vector of first partial derivatives with respect to the independent variables. Calculations of the results of operators or functions applied to GRADIENT variables are carried out according to the well-known rules for evaluation and differentiation of sums, differences, products, and so on. Since the differentiation is performed at compile time, the code produced is comparable in compactness and execution time to that obtained if numerical approximations are used for derivatives, and the theoretical and practical problems associated with numerical differentiation are avoided. PASCAL-SC source code is given for the necessary operators and standard functions, and it is shown how to prepare code for arbitrary differentiable functions to add to the library if desired. The effectiveness of the use of type GRADIENT is shown by an example of the solution of a system of nonlinear equations by Newton's method.

AMS (MOS) Subject Classifications: 65-04, 65G05, 65H10, 65K10, 68-04, 68C20

Key Words: Scientific computation, automatic differentiation, gradients, Jacobians, sensitivity analysis, error analysis, optimization, nonlinear systems of equations, PASCAL-SC

Work Unit Number 3 (Numerical Analysis and Computer Science)

Sponsored by the United States Army under Contract No. DAAG29-80-C-0041.

SIGNIFICANCE AND EXPLANATION

In scientific computation, there is often need for the derivatives as well as the values of functions defined by computer programs. Examples include calculations of sensitivity and error analysis, unconstrained and constrained optimization, and the solutions of nonlinear systems of equations. Modern computer languages permit user-defined operators and data types, so the well-known rules for differentiation can be automated. Here, this is done in the language PASCAL-SC (PASCAL for Scientific Computation), which also offers the advantages of extremely accurate floating-point arithmetic operations for real and complex numbers and intervals, and vectors and matrices over these numerical data types. For the purpose of differentiation, a data type called GRADIENT is introduced. A datum of type GRADIENT is a record of the form $(f(x),(f_1(x),\ldots,f_n(x)))$, i.e., consisting of the value at some point $x = (x_1,\ldots,x_n)$ of some function f of n variables, together with the values at x of its first n partial derivatives $f_i = \partial f/\partial x_i$, $i = 1,\ldots,n$. This requires specification of all allowable operations involving this data type. For example, the multiplication operator * is extended to the variables f and g of type GRADIENT by the prescription $f*g = (f(x)*g(x),(f(x)*g_1(x)+f_1(x)*g(x),\ldots,f(x)*g_n(x)+f_n(x)*g(x)))$. That is about all there is to it: The rules for evaluation and differentiation are applied to expressions involving GRADIENT variables to obtain the correct value and gradient vector for the result. Since the differentiation is done at compile time, the code produced is compact and efficient, and the practical and theoretical difficulties associated with numerical differentiation are avoided. As shown by an example, numerical differentiation is inaccurate and unstable even in favorable cases, and its use can render the results of a program meaningless without the user being aware of the loss of significance. Thus, when software for analytic differentiation is available, it is foolish and perhaps even dangerous to use numerical approximations to derivatives.

PASCAL-SC source code is given for the necessary operators and basic standard functions. It is shown how other functions can be added easily to the library, if one knows rules for their differentiation or how to express them in terms of differentiable functions. An example of the effectiveness of the use of type GRADIENT is its use in the solution of nonlinear systems of equations by Newton's method, which is illustrated by a specific calculation.

The responsibility for the wording and views expressed in this descriptive summary lies with MRC, and not with the author of this report.

DIFFERENTIATION IN PASCAL-SC: TYPE GRADIENT

L. B. Rall

1. Automatic differentiation. In scientific computation, there is often need for the derivatives as well as the values of the functions being computed. For problems of even moderate size, however, it is usually not feasible to differentiate all expressions appearing in the program by hand; this time-consuming process can also result in additional errors which have to be tracked down and eliminated from the final code. The alternatives are then to resort to the use of inaccurate numerical differentiation, based on difference quotients, or to have software which will enable the evaluation of the required derivatives automatically in the course of the computation. The latter approach is possible because differentiation proceeds according to fixed rules, and thus can be automated [7], [12], [13]. The implementation of this method in a modern scientific computing language is presented here; a brief comparison of the results of analytic with numerical differentiation will also be given.

In order to make an efficient implementation of the method of automatic analytic differentiation, a language is required in which the user can defined appropriate *data types*, as in ordinary PASCAL, and corresponding *operators* on these types, which is a feature of ALGOL-68, for example [15]. The scientific computing language known as PASCAL-SC [1], [19] has both of these features, and in addition, highly accurate arithmetic based on a general theory [9] for real and complex numbers and intervals, as well as vectors and matrices over these numerical types. The latter capabilities are essential for accurate scientific computation, but do not enter explicitly into the following discussion. Hence, the techniques discussed below are not limited to

---

Sponsored by the United States Army under Contract No. DAAG29-80-C-0041.

PASCAL-SC, but can be adapted readily to any language in which the user can introduce data types and operators between them. However, the accuracy and flexibility of PASCAL-SC make it the language of choice for scientific computation.

Since the implementation method described here performs analytic differentiation at *compile time*, the machine code produced is of the same order of efficiency as if expressions for the derivatives were given in the source code or, as will be seen, as if numerical differentiation by difference quotients is used.

2. Mathematical preliminaries. The computation of the value $f:=f(x_1,\ldots,x_n)$ of a function of n real variables $x_1,\ldots,x_n$ is done on a computer by a sequence of arithmetic operations and the evaluation of standard (or library) functions, for which subroutines are available. From a mathematical point of view, this means that f is the *composition* of a finite number of functions $f_1,\ldots,f_m$, that is,

$$f := f_1 \circ f_2 \circ \ldots \circ f_m, \tag{2.1}$$

where f_k is in general a function of $x_1,\ldots,x_n$ and $f_{k+1},\ldots,f_m$. If, at the current value of $x:=(x_1,\ldots,x_n)$, each f_k is differentiable with respect to its arguments, then f is differentiable at x, and f'(x) is obtained by the *chain rule* of differential calculus,

$$f'(x) := f_1'(x^{(m-1)}) \cdot \ldots \cdot f_{m-1}'(x^{(1)}) \cdot f'(x), \tag{2.2}$$

where the ' denotes Fréchet differentiation and the · matrix multiplication [13], and

$$x^{(k)} := f^{(k)}(x), \quad f^{(k)} := f_{k+1} \circ \ldots \circ f_m, \quad k := 1,\ldots,m-1. \tag{2.3}$$

If f'(x) exists, then it is represented by the *gradient vector*

$$\nabla f(x) = \left(\frac{\partial f(x)}{\partial x_1}, \frac{\partial f(x)}{\partial x_2}, \ldots, \frac{\partial f(x)}{\partial x_n}\right)^T; \tag{2.4}$$

for details about differentiation in vector spaces, see [11], for example.

As explained in [13], once software is in place for the calculation of f_k and

f'_k, then the (gradient) *vector* $\nabla f(x)$ can be obtained along with the *value* $f(x)$ at x for which f is differentiable. This software is produced by the application of *rules* to the expressions defining f: The rules for *evaluation* of expressions give the code for $f(x)$, while the rules for *differentiation* give the corresponding code for $\nabla f(x)$. From a practical standpoint, all this means is that the result of each of the operations or function evaluations involved in the computation of f will be the intermediate value of f, and the corresponding intermediate values of the partial derivatives of f with respect to $x_1,\ldots,x_n$. Since the compiler is informed of the rules for differentiation of arithmetic operations and library functions, the machine code is built in the same sequential fashion as just for the evaluation of $f(x)$ only. The implementation of differentiation discussed here is for a serial processor; as pointed out in [13], analytic differentiation can be done even more efficiently in a parallel environment.

To see what will actually be computed in mathematical notation, suppose that f is a function of only two independent variables y,z. (The concepts of independent and dependent variables will be defined precisely later.) Then, if

(2.5) $\qquad f := g(y,z),$

where $g(y,z)$ denotes an expression containing only the variables y,z, then the components of $\nabla f(y,z)$ are of course $\partial f/\partial y$ and $\partial f/\partial z$ evaluated at the current values of the independent variables. It can happen that the independent variables do not occur explicitly in the expression defining f, for example, suppose that

(2.6) $\qquad f := g(u,v),$

where $\nabla u(y,z)$ and $\nabla v(y,z)$ are known. Then, as usual, one gets

(2.7) $$\frac{\partial f}{\partial y} = \frac{\partial g}{\partial u}\cdot\frac{\partial u}{\partial y} + \frac{\partial g}{\partial v}\cdot\frac{\partial v}{\partial y}, \quad \frac{\partial f}{\partial z} = \frac{\partial g}{\partial u}\cdot\frac{\partial u}{\partial z} + \frac{\partial g}{\partial v}\cdot\frac{\partial v}{\partial z}.$$

If one or more independent variables also appear, such as in

(2.8) $\qquad f := g(y,u,v),$

then the first component of ∇f is

$$\frac{\partial f}{\partial y} = \frac{\partial g}{\partial y} + \frac{\partial g}{\partial u}\frac{\partial u}{\partial y} + \frac{\partial g}{\partial v}\frac{\partial v}{\partial y}, \tag{2.9}$$

and so on. Formula (2.9) is sometimes called a *semi-total* partial derivative, since the independent variable y occurs explicitly as well as implicitly on the right side of (2.8) [16]. The important thing is that the gradients of all variables on the right side of an assignment statement are known; the gradient of the variable on the left side can then be calculated in terms of their components.

3. A critique of numerical differentiation. Before going on to the actual implementation of automatic differentiation under consideration, a brief examination will be made of the attempt to approximate partial derivatives of a function f of n variables by means of difference quotients

$$\Delta_k^h(f,x) = \frac{f(x + he^{(k)}) - f(x)}{h} \approx \frac{\partial f(x)}{\partial x_k}, \tag{3.1}$$

where $e^{(k)}$ denotes the kth unit vector. A simple analysis [14] shows that this is an inaccurate and unstable process, even for functions of one variable, unless techniques of the differential calculus are used. The user of difference quotients is faced with the traditional dilemma facing solvers of ill-posed problems: If h is too large, then the difference quotient is a poor approximation to the derivative (truncation error), while if h is too small, then significant digits are lost in the numerator since f(x) and $f(x + he^{(k)})$ agree in many of the finite number of places to which they are calculated, because differentiability of f implies its continuity. Thus, as h becomes small, the user of difference quotients is led down the garden path, losing one significant digit after another, until one arrives in a Cloudcukooland in which the difference quotient is 0 for all h sufficiently small, regardless of the value of the derivative sought.

This situation creates a serious problem, which is to find the optimal value of h in the sense that as much accuracy as possible is attained in the approximation

of the derivative, and the loss of significant digits is minimized. An analysis by Dennis and Schnabel [3] indicates the following resolution: If f(x) can be computed with relative error τ, then $h = \sqrt{\tau}$ is a suitable choice in the sense that the difference quotient approximates the derivative to order τ, while retaining about half the number of significant digits in f(x), assuming the function and the derivative being computed are roughly the same order of magnitude. To see how this works out, it will be applied to the simple example

$$f(x) := \sin x \ , \tag{3.2}$$

a function of a single real variable. The method of analytic differentiation presented in this paper, of course, gives the exact result

$$f'(x) := \cos x \ , \tag{3.3}$$

which will be compared with the difference quotient

$$\Delta^h(f,x) := \frac{\sin(x + h) - \sin(x)}{h} \tag{3.4}$$

for x := 1.00 and x := 1.57. The particular PASCAL-SC implementation used [18] computes with twelve-digit decimal arithmetic; one gets

$$\begin{aligned} &\text{SIN(1.00)} = \text{8.41470984808E-01}, \\ &\text{SIN(1.57)} = \text{9.99999682932E-01}. \end{aligned} \tag{3.5}$$

Since these values are guaranteed to be accurate to the twelve digits shown [18], [19], it is reasonable to take $\tau = 10^{-12}$, from which the rule cited above indicates that $h = 10^{-6}$ is optimal. Actual results are shown in Tables 3.1 and 3.2.

The computed results bear out the theory in [3] to the extent that what little accuracy is obtained is maximized for the predicted value $h = 10^{-6}$. The value of the exact derivative, however, is not only more accurate but also computed *faster* in this case, even if one also wants the value of f(x), as is usually true. The loss of significant digits in $\Delta^h(f,x)$ is large in the second case, where f(x) and f'(x) differ by several orders of magnitude. If such a sacrifice of significance occurs

h	$\Delta^h(f, 1.00)$	Significant Digits	Accurate Digits
10^{-1}	4.97363752530E-02	11	0
10^{-2}	5.36085981000E-01	9	2
10^{-3}	5.39881480000E-01	8	3
10^{-4}	5.40260230000E-01	8	4
10^{-5}	5.40298100000E-01	7	5
10^{-6}	5.40302000000E-01	6	6
10^{-7}	5.40300000000E-01	4	4
10^{-8}	5.40300000000E-01	4	4
10^{-9}	5.40000000000E-01	2	2
10^{-10}	5.40000000000E-01	2	2
10^{-11}	5.00000000000E-01	1	1
10^{-12}	0.00000000000E+00	0	0
COS(1.00)	5.40302305868E-01	12	12

Table 3.1. Comparison of $\Delta^h(f, 1.00)$ with f'(1.00).

h	$\Delta^h(f, 1.57)$	Significant Digits	Accurate Digits
10^{-1}	-4.91633312200E-02	10	0
10^{-2}	-4.20364330000E-03	8	0
10^{-3}	2.96327000000E-04	6	0
10^{-4}	7.46330000000E-04	5	1
10^{-5}	7.91300000000E-04	4	2
10^{-6}	7.96000000000E-04	3	3
10^{-7}	7.90000000000E-04	2	1
10^{-8}	8.00000000000E-04	1	1
10^{-9}	1.00000000000E-03	1	1
10^{-10}	0.00000000000E+00	0	0
COS(1.57)	7.96326710733E-04	12	12

Table 3.2. Comparison of $\Delta^h(f, 1.57)$ with f'(1.57).

often enough in the program, then the final results will be rendered meaningless. Higher-order schemes for numerical differentiation [10] suffer from the same problems involving cancellation of significant digits, and are additionally slower to compute than the simple difference quotient (3.1). Use of analytic derivatives, obtained automatically, avoids this entire area of problems. Now that advanced languages which permit easy implementation of differentiation are available, the future of the use of difference quotients to approximate derivatives in scientific computation appears to be extremely limited. It should be kept in mind that the above remarks apply to differentiation of functions defined by computer programs. The differentiation of functions defined by *data* is a classical ill-posed problem [17], for which the mathematical theory and computational techniques are still being developed.

The results in Tables 3.1 and 3.2 also show the convenience of the accurate PASCAL-SC floating-point arithmetic [18]; the trailing zeros clearly indicate the number of significant digits lost.

4. Representation of TYPE GRADIENT. Returning to analytic differentiation, the discussion above shows that it is natural to associate with each differentiable function f of n variables its *value* f(x) at a point x, and its n-dimensional *gradient vector* $\nabla f(x)$ at x. This association between a real number and a real vector makes the use of the RECORD data type of PASCAL appropriate [6]. (PASCAL-SC is an extension of PASCAL, so that concepts relating to the latter carry over directly.) The standard declaration of TYPE GRADIENT then runs as follows:

```
CONST DIM = n;
 TYPE DIMTYPE = 1..DIM;
      RVECTOR = ARRAY [DIMTYPE] OF REAL;
   GRADIENT = RECORD F: REAL; DF: RVECTOR END;
```

(4.1)

The first three lines of (4.1) are simply the declaration of TYPE RVECTOR in PASCAL-SC; the value n in lower-case is supplied by the user. Since n-dimensional

real vectors are ubiquitous in scientific computation, RVECTOR is considered to be a standard numerical data type in PASCAL-SC. Thus, by following the form (4.1) exactly, the facilities of PASCAL-SC for computation with real *n*-dimensional vectors will be at one's disposal [18].

If F is of type GRADIENT, then F.F is called its *real part*, and F.DF its *vector part*. If F represents a real function f of $x = (x_1,\ldots,x_n)$, then

$$F.F = f(x),\quad F.DF[1] = \frac{\partial f(x)}{\partial x_1},\ \ldots,\ F.DF[n] = \frac{\partial f(x)}{\partial x_n} \tag{4.2}$$

in the sense that the corresponding values will be assigned to F.F and the components of F.DF. A more precise description will be given in the following section.

5. Representations of variables and constants. In order to be able to use automatic differentiation in a sensible way, it is essential to distinguish between independent and dependent variables on the one hand, and constants on the other. For the purpose of differentiation, all variables will be of type GRADIENT, while constants, as will be seen, may be of type INTEGER, REAL, or GRADIENT. The necessary distinctions will be made in more detail below.

5.1. Independent variables. A variable V of type GRADIENT is said to be the K*th independent variable* if the Kth unit vector $e^{(K)}$ is assigned to its vector part, that is, V.DF:=$e^{(K)}$, or, more precisely, if

$$V.DF[I]:=0 \text{ for } I \neq K;\ V.DF[K]:=1. \tag{5.1}$$

The user of the differentiation software described here is free to name and order independent variables in an arbitrary fashion, subject to the ordinary limitations of PASCAL [6]. For example, if X,Y,Z denote respectively the first, second, and third independent variables, then

$$\begin{aligned} &X.DF[1]:=1,\ X.DF[2]:=0,\ X.DF[3]:=0,\\ &Y.DF[1]:=0,\ Y.DF[2]:=1,\ Y.DF[3]:=0,\\ &Z.DF[1]:=0,\ Z.DF[2]:=0,\ Z.DF[3]:=1, \end{aligned} \tag{5.2}$$

define their vector parts X.DF, Y.DF, and Z.DF, respectively. If independent variables appear in assignment statements, it should be only on the right side. An assignment to the vector part of the Kth independent variable of anything but the Kth unit vector makes that variable dependent (see below), and introduces a new, but perhaps unspecified Kth independent variable, with respect to which subsequent partial derivatives will be computed. Since the rules for partial differentiation are applied rapidly and accurately, but mindlessly, during the course of the compilation, the user should avoid introducing errors of this type unless a change of variable is actually intended. In the latter case, assignments to former independent variables of their values in terms of new ones performs the change of variables automatically, given the corresponding expressions and subroutines. Of course, it will often happen in the course of the computation that assignments will be made to the value part V.F of an independent variable V.

Another possible source of difficulty in setting up independent variables would be the assignment of the same unit vector to the vector parts of two supposedly independent variables, say V and W, that is, V.DF = W.DF. This means that V and W would be considered to be the same variable for the purpose of differentiation, but with possibly different values V.F and W.F. This might be used as a sneaky way of assigning several values to the Kth independent variable, but is not recommended, since the same purpose can be served by a straightforward method which avoids the possibility of confusion and perhaps well-concealed errors.

5.2. Dependent variables. Variables appearing on the left side of assignment statements are, of course, *dependent* on the variables appearing in the expressions on the right side, and thus ultimately on the set of independent variables chosen by the user. A dependent variable F depends on the Kth independent variable or not according as F.DF[K] $\neq$ 0 or F.DF[K] = 0.

5.3. Constants. A "variable" of type GRADIENT is said to be a *constant* if its vector part is the n-dimensional zero vector 0 = (0,...,0), that is, C is a

constant if

(5.3) C.DF[I] = 0, I = 1,...,n.

Constants of type GRADIENT can be produced in the computation by direct assignment, or by expressions such as F:=0*X, F:=X/X, F:=X-X, and so on. It is also permissible to have variables or literal constants of type INTEGER or REAL in GRADIENT expressions; such quantities will be treated as constants for the purpose of differentiation. Thus, an assignment statement such as

(5.4) F := (K + X)**(4.4 + Y) - R - 3;

where K is of type INTEGER, R is REAL, and F,X,Y are GRADIENT, is allowed. This makes it easy to handle expressions involving real or integer-valued parameters. If X and Y are independent variables, then only $\partial F/\partial X$ and $\partial F/\partial Y$ will be altered in F.DF by the assignment (5.4), in addition to the value part F.F of F.

6. Definition of operators for type GRADIENT. In order to compute with variables of type GRADIENT, the arithmetic operators and the standard and other functions needed must be defined in order to conform to the rules for evaluation and differentiation of real functions. Arithmetic operations +,-,*,/,** will be considered in this section, and the standard and other functions in the next. In what follows, K, R, RA, RB, G, GA,GB will denote generic variables of the following types:

(6.1) VAR K: INTEGER; R,RA,RB: REAL; G,GA,GB: GRADIENT;

the operator extension in PASCAL-SC [1] will be employed to obtain the desired results. The source code for the arithmetic operators is given in Appendix A.

6.1. Addition (+). A total of six definitions of the addition operators are required: One each for the various combinations

(6.2) +G, K+G, G+K, R+G, G+R, GA+GB.

For example, the unary addition (identity) operator denoted by +G is declared by

```
                OPERATOR + (G: GRADIENT) RES: GRADIENT;
(6.3)
                    BEGIN RES:=G END;
```

according to the rules of PASCAL-SC [1]. Similarly, the assignments F:=K+G, F:=G+K, F:=R+G, F:=G+R alter only the value part of F; one has respectively F.F:=K+G.F, F.F:=G.F+K, F.F:=R+G.F, F.F:=G+R, while F.DF:=G.DF in each case. Finally, corresponding to GA+GB, one has the operator declaration

```
                OPERATOR + (GA: GRADIENT;GB: GRADIENT) RES: GRADIENT;
                    VAR U: GRADIENT;I: DIMTYPE;
                    BEGIN U.F:=GA.F+GB.F;FOR I:=1 TO DIM DO
(6.4)
                          U.DF[I]:=GA.DF[I]+GB.DF[I];
                          RES:=U
                    END;
```

since the derivative of the sum is the sum of the derivatives. In mathematical notation, if u,v denote generic functions, c a constant, and x an independent variable, the operator declarations simply implement the rules for differentiation

$$(6.5) \qquad \frac{\partial}{\partial x}(+u) = \frac{\partial}{\partial x}(c + u) = \frac{\partial}{\partial x}(u + c) = \frac{\partial u}{\partial x}\ , \quad \frac{\partial}{\partial x}(u+v) = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial x}$$

for the vector parts of dependent gradient variables.

6.2. Subtraction (-). Once again, operators are required for the combinations

(6.6) -G, K-G, G-K, R-G, G-R, GA-GB

of GRADIENT, INTEGER, and REAL types. For example, unary subtraction -G (sign changing), is accomplished in GRADIENT statements by the operator declared by

```
                OPERATOR - (G: GRADIENT) RES: GRADIENT;
                    VAR U: GRADIENT;I: DIMTYPE;
(6.7)
                    BEGIN U.F:=-G.F;FOR I:=1 TO DIM DO U.DF[I]:=-G.DF[I];RES:=U
                    END;
```

Here, the rules corresponding to (6.5) are

$$(6.8)\qquad \frac{\partial}{\partial x}(-u) = \frac{\partial}{\partial x}(c - u) = -\frac{\partial u}{\partial x},\quad \frac{\partial}{\partial x}(u - c) = \frac{\partial u}{\partial x},\quad \frac{\partial}{\partial x}(u - v) = \frac{\partial u}{\partial x} - \frac{\partial v}{\partial x},$$

which, in symbolic form, give the assignments

```
(K-G).F:=K-G.F;   (K-G).DF:=-G.DF;   (R-G).F:=R-G.F;   (R-G).DF:=-G.DF;
(G-K).F:=G.F-K;   (G-K).DF:=G.DF;    (G-R).F:=G.F-R;   (G-R).DF:=G.DF;
(GA-GB).F:=GA.F-GB.F;   (GA-GB).DF:=GA.DF-GB.DF;
```

(6.9)

where the vector operations are understood to be performed componentwise, as in usual vector algebra.

6.3. Multiplication (*). Here, operators must be defined for the combinations

$$(6.10)\qquad K*G,\quad G*K,\quad R*G,\quad G*R,\quad GA*GB.$$

The rules for differentiation of products are, of course,

$$(6.11)\qquad \frac{\partial}{\partial x}(c\cdot u) = \frac{\partial}{\partial x}(u\cdot c) = c\cdot\frac{\partial u}{\partial x},\quad \frac{\partial}{\partial x}(u\cdot v) = u\cdot\frac{\partial v}{\partial x} + \frac{\partial u}{\partial x}\cdot v,$$

which lead to the symbolic assignments

```
(K*G).F:=K*G.F;   (K*G).DF:=K*G.DF;   (R*G).F:=R*G.F;   (R*G).DF:=R*G.DF;
(G*K).F:=G.F*K;   (G*K).DF:=G.DF*K;   (G*R).F:=G.F*R;   (G*R).DF:=G.DF*R;
(GA*GB).F:=GA.F*GB.F;   (GA*GB).DF:=GA.F*GB.DF+GA.DF*GB.F;
```

(6.12)

the detailed code is given in Appendix A.

6.4. Division (/). Here, as for multiplication, the combinations for which operators are needed are

$$(6.13)\qquad K/G,\quad G/K,\quad R/G,\quad G/R,\quad GA/GB.$$

The rules for differentiation of quotients are implemented in the forms

$$(6.14)\qquad \frac{\partial}{\partial x}(c/u) = -(c/u)\cdot\frac{\partial u}{\partial x}/u,\quad \frac{\partial}{\partial x}(u/c) = \frac{\partial u}{\partial x}/c,\quad \frac{\partial}{\partial x}(u/v) = \frac{\partial u}{\partial x}/v - \frac{\partial v}{\partial x}\cdot(u/v)/v.$$

Letting C stand for K or R, these correspond to the assignments

```
(C/G).F:=C/G.F;  (C/G).DF:=-G.DF*(C/G).F/G.F;
(G/C).F:=G.F/C;  (G/C).DF:=G.DF/K;
(GA/GB).F:=GA.F/GB.F;  (GA/GB).DF:=GA.DF/GB.F-GB.DF*(GA/GB).F/GB.F;
```

(6.15)

in symbolic form. Explicitly, source code for the operator / between gradient operands GA,GB is:

```
OPERATOR / (GA: GRADIENT;GB: GRADIENT) RES: GRADIENT;
    VAR U: GRADIENT;I: DIMTYPE;
    BEGIN U.F:=GA.F/GB.F;FOR I:=1 TO DIM DO
       BEGIN IF GA.DF[I] = 0 THEN U.DF[I]:=0 ELSE U.DF[I]:=GA.DF[I]/GB.F;
             IF GB.DF[I] <> 0 THEN U.DF[I]:=U.DF[I]-GB.DF[I]*U.F/GB.F
       END;
       RES:=U
    END:
```

(6.16)

as given in Appendix A. Attempted division by zero produces an error interrupt.

6.5. Power (**). The power operator defined by

$$U^{**}V = U^{V} \tag{6.17}$$

is not standard in PASCAL or PASCAL-SC. Therefore, this operator is also introduced for R**K, a REAL raised to an INTEGER power, and RA**RB for RA,RB of type REAL. An algorithm which is recommended for this purpose [2] is based on

$$U^{**}V := 10^{V*LOG10(U)}, \tag{6.18}$$

since decimal arithmetic is being used. However, experiments with integer exponents showed that more accuracy was obtained in this case by use of the method from [13] of repeated squaring of the base, as employed in the original differentiation software written by Reiter [5]. Specifically, assume that K is a non-negative integer, and

(6.19) $$K = \varepsilon_0 + \varepsilon_1 \cdot 2 + \varepsilon_2 \cdot 2^2 + \ldots + \varepsilon_m \cdot 2^m,$$

where $\varepsilon_m = 1$ and $\varepsilon_i = 0$ or 1, $i = 0,1,\ldots,m-1$. Then,

(6.20) $$R^K = \prod_{i=0}^{m} R^{\varepsilon_i \cdot 2^i} = \prod_{i=0}^{m} R_i^{\varepsilon_i},$$

where

(6.21) $$R_0 = R, \quad R_i = R_{i-1}^2, \quad i = 1,\ldots,m.$$

Thus, each factor R_i is formed by squaring the previous R_{i-1}, and is multiplied into the product (6.20) or not according as $\varepsilon_i = 1$ or 0. Hence the name "repeated squaring" for this algorithm. For negative K, the reciprocal of the result for positive K is taken. As pointed out in [13], one takes $R^0 = 1$ for $R \neq 0$, $0^K = 0$ for $K > 0$, while 0^K for $K \leq 0$ leads to an error interrupt due to an attempted division by 0. The case $R^1 = R$ is also handled separately in the source code, which is

(6.22)
```
OPERATOR ** (R: REAL;K: INTEGER) RES: REAL;
    VAR L: INTEGER;U: REAL;
    BEGIN IF K <= 0 THEN U:=1/R;IF K = 0 THEN U:=1
       ELSE IF K = 1 THEN U:=R
          ELSE BEGIN L:=ABS(K);U:=1;REPEAT IF L MOD 2 = 1
                    THEN U:=R*U;L:=L DIV 2; IF L <>0
                    THEN R:=R*R UNTIL L = 0;
                    IF K <0 THEN U:=1/U
               END;
       RES:=U
    END;
```

The computation of RA**RB for a REAL exponent RB is based on the same algorithm; one computes RA**K for the INTEGER exponent K:=TRUNC(RB). Then, $-1 < RB - K < 1$, and

(6.23) RA**RB:=(RA**K)*EXP((RB-K)*LN(RA));

in which natural logarithms are used. Negative bases RA with nonintegral exponents will lead to an error interrupt, since the logarithm function will not accept negative arguments; an attempt to compute 0^{RB} for RB ≤ 0 will similarly be wrecked by a division by zero error interrupt.

Once operators for integral and real powers of real arguments are in place, the corresponding operators for gradient variables can be derived from the rules for differentiation,

$$(6.24) \qquad \frac{\partial}{\partial x}(u^c) = c \cdot u^{c-1} \cdot \frac{\partial u}{\partial x}, \quad \frac{\partial}{\partial x}(c^u) = c^u \cdot \ln(c) \cdot \frac{\partial u}{\partial x},$$

and

$$(6.25) \qquad \frac{\partial}{\partial x}(u^v) = v \cdot u^{v-1} \cdot \frac{\partial u}{\partial x} + u^v \cdot \ln(v) \cdot \frac{\partial v}{\partial x}.$$

The power operator ** is thus available for the combinations

(6.26) R**K, RA**RB, K**G, G**K, R**G, G**R, GA**GB.

For the forms involving GRADIENT expressions, the symbolic assignments are:

```
(G**C).F:=G.F**C;  (G**C).DF:=C*((G**C).F/G.F)*G.DF;
(C**G).F:=C**G.F;  (C**G).DF:=(C**G.F)*LN(C)*G.DF;
(GA**GB).F:=GA.F**GB.F;  (GA**GB).DF:=(GA.F**GB).DF+(GA**GB.F).DF;
```

(6.27)

once again, all operations involving the vector parts .DF of the gradient variables are to be interpreted componentwise.

6.6. Priorities of Operators. The introduction of type GRADIENT thus requires the definition of 29 operators; six each for addition and subtraction, five each for multiplication and division, and the seven power operators, including the two for raising a REAL base to an INTEGER or REAL power. The latter can also be used in ordinary numerical computation. The priorities of these gradient operators

and the two real operators defined above are as follows:

1st priority: Unary addition and subtraction ±;

2nd priority: Multiplication, division, and power *,/,**;

3rd priority: Binary addition and subtraction +,-.

As usual, parentheses are introduced to achieve the desired order of operations. Users familiar with languages in which ** has a higher priority than *,/ should be particularly careful to make their desires explicit. For example, 2*3**4 means $6^4 = 1,296$, not $2 \cdot 3^4 = 162$.

7. Standard gradient functions. The present implementation for type GRADIENT includes the following standard function: Absolute value, and the six standard functions for type REAL available in the PASCAL-SC compiler [18] (square root, exponential (base e), natural logarithm, arctangent, sine, and cosine). All standard gradient functions have names which begin with G: GABS, GSQRT, GEXP, GLN, GARCTAN, GSIN, and GCOS. Thus, to obtain the value and the gradient vector for the function

(7.1) $$f(x,y) = (xy + \sin x + 4)(3y^2 + 6),$$

[11], [12], [13], the corresponding GRADIENT assignment statement is

(7.2)
```
F:=(X*Y + GSIN(X) + 4)*(3*(Y**2) + 6);
```

keeping in mind the equal priority of * and **. (The software described in this paper was tested first on (7.2), for historical reasons.) The minor nuisance of having to write GSIN(X) instead of SIN(X) is balanced by the fact that the former makes it clear that (7.2) is a GRADIENT assignment statement, and GRADIENT arguments are expected in the expression on the right, at least for X in the sine function and thus also in the product X*Y.

For the absolute value, one has

(7.3) $$\frac{\partial}{\partial x}|u| = -\frac{\partial u}{\partial x} \text{ if } u < 0, \quad \frac{\partial}{\partial x}|u| = \frac{\partial u}{\partial x} \text{ if } u > 0, \quad \frac{\partial}{\partial x}|u| \text{ undefined if } u = 0.$$

In source code (see Appendix B), this becomes:

```
FUNCTION GABS(G: GRADIENT): GRADIENT;
    VAR I: DIMTYPE;M: REAL;U: GRADIENT;
    BEGIN U.F:=ABS(G.F);M:=G.F/U.F;FOR I:= 1 TO DIM DO IF G.DF[I]=0 THEN
        U.DF[I]:=0 ELSE U.DF[I]:=M*G.DF[I];
        GABS:=U
    END;
```

(7.4)

the nondifferentiablity of this function at zero will be indicated, if attempted, by a division by zero error interrupt.

The remaining standard functions follow the pattern (7.4) exactly, using their known derivatives to construct assignments to the *multiplier* (or divisor) M. In brief form, the required assignments are

```
U.F:=SQRT(G.F);  M:=2*U.F;  U.DF:=G.DF/M;  GSQRT:=U;
U.F:=EXP(G.F);  U.DF:=U.F*G.DF;  GEXP:=U;
U.F:=LN(G.F);  U.DF:=G.DF/G.F;  GLN:=U;
U.F:=ARCTAN(G.F);  M:=1+G.F*G.F;  U.DF:=G.DF/M;  GARCTAN:=U;
U.F:=SIN(G.F);  M:=COS(G.F);  U.DF:=M*G.DF;  GSIN:=U;
U.F:=COS(G.F);  M:=-SIN(G.F);  U.DF:=M*G.DF;  GCOS:=U;
```

(7.5)

for the respective standard functions. An explicit assignment to M was unnecessary in the case of the exponential and logarithmic functions.

8. User-defined functions. The user, of course, is free to add functions (or procedures) for type GRADIENT, as long as the rules for differentiation of the results are known explicitly. In the case of functions of a single variable, the code (7.4) can act as a template for the required code. If the real function f is known as FUNC, and its derivative f' is computed by DFUNC, then the gradient function GFUNC requires the assignments

```
U.F:=FUNC(G.F);  M:=DFUNC(G.F);  U.DF:=M*G.DF;  GFUNC:=U;
```

(8.1)

which correspond to the rule for partial differentiation

$$\frac{\partial}{\partial x} f(u) = f'(u) \cdot \frac{\partial u}{\partial x}. \tag{8.2}$$

For example, suppose that the cosecant and its derivatives are needed. Then, following the pattern of (7.4), the user can introduce the function GCSC by the code:

```
FUNCTION GCSC(G: GRADIENT): GRADIENT;
    VAR I: DIMTYPE;M: REAL;U: GRADIENT;
    BEGIN U.F:=1/SIN(G.F);M:=-COS(G.F)*U.F*U.F;FOR I:=1 TO DIM DO
          IF G.DF[I] = 0 THEN U.DF[I]:=0 ELSE U.DF[I]:=M*G.DF[I];
          GCSC:=U
    END;
```
(8.3)

however, since the function in question can be expressed easily in terms of available gradient functions and operators, a more compact code is

```
FUNCTION GCSC(G: GRADIENT): GRADIENT;
    BEGIN GCSC:=1/GSIN(G) END;
```
(8.4)

The coding of gradient functions and operators can also be simplified by using the operators and functions for vector and matrix algebra available in the PASCAL-SC library [18]. This approach is particularly useful if it is desired to precompile the gradient operators and functions, and store the relocatable code generated in an external library. This cannot be done for the source code given in the appendices, since most of these routines contain the global variable DIM, which can be removed if source code for vector operations is brought into the program. In vector form, (8.3) becomes

```
FUNCTION GCSC(G: GRADIENT): GRADIENT;
    VAR M: REAL;U: GRADIENT;
    BEGIN U.F:=1/SIN(G.F);M=-COS(G.F)*U.F*U.F;U.DF:=M*G.DF;
          GCSC:=U
    END;
```
(8.5)

with no global variables. In all cases, the assignment U.DF:=M*G.DF should be arranged so that the multiplication of the RVECTOR G.DF by the REAL M is from the left.

9. An example: Newton's method for the solution of nonlinear systems of equations. Type GRADIENT has many applications in scientific computing, for example, to sensitivity and error analysis, optimization, and the solution of systems of nonlinear equations [13]. In order to apply Newton's method to the numerical solution of the system of equations,

$$\begin{aligned} &f_1(x_1,x_2,\ldots,x_n) = 0,\\ &f_2(x_1,x_2,\ldots,x_n) = 0,\\ &\ldots \quad \ldots \quad \ldots \quad \ldots\\ &f_n(x_1,x_2,\ldots,x_n) = 0, \end{aligned} \tag{9.1}$$

the n×n *Jacobian matrix* $J = (\partial f_i/\partial x_j)$ is needed [11]. Without automatic differentiation, the labor involved in producing J from (9.1), not to speak of the possibility of introduction of errors, is prohibitive even for moderate n in the general case. However, if F1,...,FN and X1,...,XN are of type GRADIENT, then the Ith row of the Jacobian matrix J is simply FI.DF, the vector part of FI, assuming X1,...,XN are independent variables. Specifically, suppose that the PASCAL-SC type RMATRIX is declared by

```
TYPE RMATRIX = ARRAY [DIMTYPE] OF RVECTOR;
```  (9.2)

where DIMTYPE and RVECTOR are introduced as in (4.1), and the new type JACOBIAN by

```
TYPE JACOBIAN = ARRAY [DIMTYPE] OF GRADIENT;
```  (9.3)

further, suppose X,F are of type Jacobian, where X[I].DF is the Ith unit vector. Then, X[I] will be the Ith independent variable, and for

$$F[I]:=f_I(X[1],X[2],\ldots X[n]), \quad I = 1,\ldots,n, \tag{9.4}$$

the problem of solving the system (9.1) amounts to finding the solution vector with

components X[I].F,...,X[n].F such that F[I].F = 0, I = 1,...,n.

Returning to the system (9.1), let $x = (x_1,\ldots,x_n)^T$, $f(x) = (f_1(x),\ldots, f_n(x))^T$. Then, one step of Newton's method, starting from the approximation $x^{(0)}$ to the solution x of (9.1), requires the solution of the linear system of equations

$$J\cdot\delta = -f(x^{(0)}) \tag{9.5}$$

for the *correction vector* $\delta = (\delta_1,\ldots,\delta_n)^T$, from which the next approximation $x^{(1)}$ to x is obtained as

$$x^{(1)} = x^{(0)} + \delta, \tag{9.6}$$

see, for example, [11]. Suppose that one has a function

```
FUNCTION SOLVLN (N: DIMTYPE;A: RMATRIX;B: RVECTOR): RVECTOR;
```
(9.7)

which will give the solution of a system of N linear equations in N unknowns with coefficient matrix A and right-hand side B. (A very accurate procedure for the solution of linear systems called LGLP is available in PASCAL-SC [18].) The code for one step of Newton's method with X,F of type JACOBIAN and B,D of type RVECTOR, J (the Jacobian matrix), of type RMATRIX, is

```
FOR I:=1 TO DIM DO BEGIN J[I] := F[I].DF;B[I]:=-F[I].F END;
D:=SOLVLN(DIM,J,B);
FOR I:=1 TO DIM DO X[I].F:=X[I].F+D[I];
```
(9.8)

for the general case.

It is not necessary or always convenient to introduce the type JACOBIAN. For example, suppose that X,Y,Z are the independent variables of type GRADIENT (see (5.2), and the GRADIENT dependent variables F,G,H are defined by

```
F:=16*(X**4)+16*(Y**4)+Z**4-16;
G:=X**2+Y**2+Z**2-3;
H:=X**3-Y;
```
(9.9)

thus, the conditions F.F = G.F = H.F = 0 correspond to the system of equations

$$
\begin{aligned}
16x^4 + 16y^4 + z^4 - 16 &= 0,\\
x^2 + y^2 + z^2 - 3 &= 0,\\
x^3 - y &= 0,
\end{aligned}
\tag{9.10}
$$

which was chosen, again for historical reasons [4], to test the gradient Newton method program. For the assignments (9.9), the code (9.8) is expanded for B,D of type RVECTOR, J (the Jacobian matrix) of type RMATRIX, to

```
J[1]:=F.DF;J[2]:=G.DF;J[3]:=H.DF;
B[1]:=-F.F;B[2]:=-G.F;B[3]:=-H.F;
D:=SOLVLN(DIM,J,B);
X.F:=X.F+D[1];Y.F:=Y.F+D[2];Z.F:=Z.F+D[3];
```

(9.11)

the result of eight iterations, starting from X.F = Y.F = Z.F = 1, is

```
X.F = 8.77965760274E-01,     F.F = 0.00000000000E+00,
Y.F = 6.76756970518E-01,     G.F = 0.00000000000E+00,
Z.F = 1.33085541162E+00,     H.F = 0.00000000000E+00.
```

(9.12)

A summary of the complete calculation is given in Appendix C. The results agree exactly to the given number of places to the results for (9.10) obtained on a UNIVAC 1100 in double precision, using the program NEWTON [8], which also employs automatic differentiation.

10. Implementation details. The software described in this report was implemented and tested on a Zilog MCZ 1/05 microcomputer. This machine has a Z80 processor and 64 kilobytes of main storage, augmented by two 8" disk drives for single-sided, single-density hard-sectored disks. With one drive dedicated to the system disk, this gives the user about 308 kilobytes of mass storage. The PASCAL-SC compiler used was obtained from the Institute for Applied Mathematics, University of Karlsruhe, Germany, and runs under the Zilog RIO 2.06 operating system. The modest size of the machine limits one to about 20×20 matrices. The program to solve the system (9.10) required less than 5,632 bytes, showing the compactness of the software.

11. Conclusions. By an actual example, it has been shown that a modern computer language which permits user-defined operators and data types allows the implementation of automatic analytic differentiation to compute the gradient vector as well as the value of real functions of n real variables. This allows the accuracy and theoretical advantages of the use of analytically defined derivatives in actual computation [12], [13], and avoids the problems associated with the approximation of derivatives by difference quotients [3]. In addition, source code using type GRADIENT is easier to prepare and understand than if approximations to derivatives are used, and the compiled code is comparable in size and execution time to that produced in the latter case.

REFERENCES

1. G. Bohlender, K. Grüner, E. Kaucher, R. Klatte, W. Krämer, U. W. Kulisch, S. M. Rump, Ch. Ullrich, J. Wolff von Gudenberg, and W. L. Miranker. PASCAL-SC: A PASCAL for scientific computation, Research Report RC 9009, IBM Thomas J. Watson Research Center, Yorktown Heights, N.Y., 1981.

2. W. J. Cody and W. Waite. Software Manual for the Elementary Functions, Prentice-Hall, Englewood Cliffs, N.J., 1980.

3. J. E. Dennis, Jr. and R. B. Schnabel. Numerical Methods for Unconstrained Optimization and Nonlinear Equations, Prentice-Hall, Englewood Cliffs, N.J., 1982.

4. Julia H. Gray and L. B. Rall. NEWTON: A general purpose program for solving nonlinear systems, Proceedings of the 1967 Army Numerical Analysis Conference, pp. 11-59. U. S. Army Research Office, Durham, N.C., 1967.

5. Julia H. Gray and Allen Reiter. Compiler of Differential Expressions (CODEX) for the CDC 3600, Technical Summary Report No. 791, Mathematics Research Center, University of Wisconsin-Madison, 1967.

6. K. Jensen and N. Wirth. PASCAL User Manual and Report, 2nd Ed., Springer-Verlag, New York-Heidelberg-Berlin, 1974.

7. G. Kedem. Automatic differentiation of computer programs, ACM Trans. Math. Software 6, no. 2 (1980), 150-165.

8. D. Kuba and L. B. Rall. A UNIVAC 1108 program for obtaining rigorous error estimates for approximate solutions of systems of equations, Technical Summary Report No. 1168, Mathematics Research Center, University of Wisconsin-Madison, 1972.

9. U. W. Kulisch and W. L. Miranker. Computer Arithmetic in Theory and Practice, Academic Press, New York, 1981.

10. W. E. Milne. Numerical Calculus, Princeton University Press, Princeton, N.J., 1949.

11. L. B. Rall. Computational Solution of Nonlinear Operator Equations, Wiley, New York, 1969; reprinted by Krieger, Huntington, N.Y., 1979.

12. L. B. Rall. Applications of software for automatic differentiation in numerical computation, Computing, Suppl. 2 (1980), 141-156.

13. L. B. Rall. Automatic Differentiation: Techniques and Applications, Lecture Notes in Computer Science No. 120, Springer-Verlag, Berlin-Heidelberg-New York, 1981.

14. J. Barkley Rosser and Carl de Boor. Pocket Calculator Supplement for Calculus, Addison-Wesley, Reading, Massachusetts, 1979.

15. H. Rutishauser. Description of ALGOL-60, Springer-Verlag, Berlin-Heidelberg-New York, 1967.

16. E. S. and I. S. Sokolnikoff. Higher Mathematics for Engineers and Physicists, McGraw-Hill, New York, 1945.

17. A. N. Tikhonov and V. Y. Arsenin. Solutions of Ill-Posed Problems (English Translation Edited by Fritz John), Winstcn, Washington, D.C., 1977.

18. J. Wolff von Gudenberg. Gesamte Arithmetik des PASCAL-SC Rechners: Benutzerhandbuch, Institute for Applied Mathematics, University of Karlsruhe, Germany, 1981.

19. J. Wolff von Gudenberg. Einbettung allgemeiner Rechnerarithmetik in PASCAL mittels eines Operatorkonzeptes und Implementierung der Standardfunktionen mit optimaler Genauigkeit. Dissertation, University of Karlsruhe, Germany, 1980.

APPENDIX A. SOURCE CODE FOR GRADIENT OPERATORS

(File GRAD_ARITH.S)

A.1. Addition Operators (File GRAD_ADD.S)

```
OPERATOR + (G: GRADIENT) RES: GRADIENT;
      BEGIN RES:=G END;

OPERATOR + (K: INTEGER;G: GRADIENT) RES: GRADIENT;
      VAR U: GRADIENT;
      BEGIN U.F:=K+G.F;U.DF:=G.DF;RES:=U END;

OPERATOR + (G: GRADIENT;K: INTEGER) RES: GRADIENT;
      VAR U: GRADIENT;
      BEGIN U.F:=G.F+K;U.DF:=G.DF:RES:=U END;

OPERATOR + (R: REAL;G: GRADIENT) RES: GRADIENT;
      VAR U: GRADIENT;
      BEGIN U.F:=R+G.F;U.DF:=G.DF;RES:=U END;

OPERATOR + (G: GRADIENT;R: REAL) RES: GRADIENT;
      VAR U: GRADIENT;
      BEGIN U.F:=G.F+R;U.DF:=G.DF;RES:=U END;

OPERATOR + (GA: GRADIENT;GB: GRADIENT) RES: GRADIENT;
      VAR U: GRADIENT;I: DIMTYPE;
      BEGIN U.F:=GA.F+GB.F;FOR I:=1 TO DIM DO
            U.DF[I]:=GA.DF[I]+GB.DF[I];
            RES:=U
      END;
```

A.2. Subtraction Operators (File GRAD_SUB.S)

```
OPERATOR - (G: GRADIENT) RES: GRADIENT;
      VAR U: GRADIENT;I: DIMTYPE;
      BEGIN U.F:=-G.F;FOR I:=1 TO DIM DO U.DF[I]:=-G.DF[I];RES:=U
      END;
```

```
OPERATOR - (K: INTEGER;G: GRADIENT) RES: GRADIENT;
     VAR U: GRADIENT;I: DIMTYPE;
     BEGIN U.F:=K-G.F;FOR I:=1 TO DIM DO U.DF[I]:=-G.DF[I];RES:=U END;

OPERATOR - (G: GRADIENT;K: INTEGER) RES: GRADIENT;
     VAR U: GRADIENT;
     BEGIN U.F:=G.F-K;U.DF:=G.DF;RES:=U END;

OPERATOR - (R: REAL;G: GRADIENT) RES: GRADIENT;
     VAR U: GRADIENT;I: DIMTYPE;
     BEGIN U.F:=R-G.F:FOR I:=1 TO DIM DO U.DF[I]:=-G.DF[I];RES:=U END;

OPERATOR - (G: GRADIENT;R: REAL) RES: GRADIENT;
     VAR U: GRADIENT;
     BEGIN U.F:=G.F-R;U.DF:=G.DF;RES:=U END;

OPERATOR - (GA: GRADIENT;GB: GRADIENT) RES: GRADIENT;
     VAR U: GRADIENT;I: DIMTYPE;
     BEGIN U.F:=GA.F-GB.F;FOR I:=1 TO DIM DO
          U.DF[I]:=GA.DF[I]-GB.DF[I];
          RES:=U;
     END;
```

A.3. Multiplication Operators (File GRAD_MUL.S)

```
OPERATOR * (K: INTEGER;G: GRADIENT) RES: GRADIENT;
     VAR U: GRADIENT;I: DIMTYPE;
     BEGIN U.F:=K*G.F;FOR I:= 1 TO DIM DO U.DF[I]:=K*G.DF[I];RES:=U END;

OPERATOR * (G: GRADIENT;K: INTEGER) RES: GRADIENT;
     VAR U: GRADIENT;I: DIMTYPE;
     BEGIN U.F:=G.F*K;FOR I:=1 TO DIM DO U.DF[I]:=G.DF[I]*K;RES:=U END;

OPERATOR * (R: REAL;G: GRADIENT) RES: GRADIENT;
     VAR U: GRADIENT;I: DIMTYPE;
     BEGIN U.F:=R*G.F;FOR I:=1 TO DIM DO U.DF[I]:=R*G.DF[I];RES:=U END;
```

```
OPERATOR * (G: GRADIENT;R: REAL) RES: GRADIENT;
     VAR U: GRADIENT;I: DIMTYPE;
     BEGIN U.F:=G.F*R;FOR I:=1 TO DIM DO U.DF[I]:=G.DF[I]*R;RES:=U END;

OPERATOR * (GA: GRADIENT;GB: GRADIENT) RES: GRADIENT;
     VAR U: GRADIENT;I: DIMTYPE;
     BEGIN U.F:=GA.F*GB.F;FOR I:=1 TO DIM DO
          U.DF[I]:=GA.F*GB.DF[I]+GA.DF[I]*GB.F;
          RES:=U
     END;
```

A.4. Division Operators (File GRAD_DIV.S)

```
OPERATOR / (K: INTEGER;G: GRADIENT) RES: GRADIENT;
     VAR U: GRADIENT;I: DIMTYPE;
     BEGIN U.F:=K/G.F;FOR I:=1 TO DIM DO
          IF G.DF[I] = 0 THEN U.DF[I]:=0 ELSE U.DF[I]:=-G.DF[I]*U.F/G.F;
          RES:=U
     END;

OPERATOR / (G: GRADIENT;K: INTEGER) RES: GRADIENT;
     VAR U: GRADIENT;I: DIMTYPE;
     BEGIN U.F:=G.F/K;FOR I:=1 TO DIM DO U.DF[I]:=G.DF[I]/K;RES:=U END;

OPERATOR / (R: REAL;G: GRADIENT) RES: GRADIENT;
     VAR U: GRADIENT;I: DIMTYPE;
     BEGIN U.F:=R/G.F;FOR I:=1 TO DIM DO
          IF G.DF[I] = 0 THEN U.DF[I]:=0 ELSE U.DF[I]:=-G.DF[I]*U.F/G.F;
          RES:=U
     END;

OPERATOR / (G: GRADIENT;R: REAL) RES: GRADIENT;
     VAR U: GRADIENT;I: DIMTYPE;
     BEGIN U.F:=G.F/R;FOR I:=1 TO DIM DO U.DF[I]:=G.DF[I]/R;RES:=U END;
```

```
OPERATOR / (GA: GRADIENT;GB: GRADIENT) RES: GRADIENT;
    VAR U: GRADIENT;I: DIMTYPE;
    BEGIN U.F:=GA.F/GB.F;FOR I:=1 TO DIM DO
       BEGIN IF GA.DF[I] = 0 THEN U.DF[I]:=0 ELSE U.DF[I]:=GA.DF[I]/GB.F;
             IF GB.DF[I] <> 0 THEN U.DF[I]:=U.DF[I]-GB.DF[I]*U.F/GB.F
       END;
       RES:=U
    END;
```

A.5. Power Operators (File GRAD_POW.S)

```
OPERATOR ** (R: REAL;K: INTEGER) RES: REAL;
    VAR L: INTEGER;U: REAL;
    BEGIN IF K <= 0 THEN U:=1/R;
          IF K = 0 THEN U:=1;
                ELSE IF K = 1 THEN U:=R
                     ELSE BEGIN L:=ABS(K);U:=1;REPEAT IF L MOD 2 = 1
                                THEN U:=R*U;L:=L DIV 2; IF L <> 0
                                THEN R:=R*R UNTIL L = 0;
                                IF K < 0 THEN U:=1/U
                          END;
          RES:=U
    END;

OPERATOR ** (RA: REAL;RB: REAL) RES: REAL;
    VAR K: INTEGER;U: REAL;
    BEGIN IF RA = 1 THEN U:=1 ELSE IF (RA=0) AND (RB > 0) THEN U:=0 ELSE
       BEGIN K:=TRUNC(RB);U:=RA**K;IF RB <> K THEN
             U:=U*EXP((RB-K)*LN(RA));
       END;
       RES:=U
    END;
```

```
OPERATOR ** (G: GRADIENT;K: INTEGER) RES: GRADIENT;
     VAR I: DIMTYPE;M: REAL;U: GRADIENT;
     BEGIN U.F:=G.F**K;IF K = 1 THEN U.DF:=G.DF ELSE
           IF K = 0 THEN FOR I:=1 TO DIM DO U.DF[I]:=0 ELSE
           IF (G.F=0) AND (K > 1) THEN FOR I:=1 TO DIM DO U.DF[I]:=0 ELSE
           BEGIN M:=K*U.F/G.F;FOR I:=1 TO DIM DO IF G.DF[I] = 0 THEN
                 U.DF[I]:=0 ELSE U.DF[I]:=M*G.DF[I]
           END;
           RES:=U
     END;

OPERATOR ** (G: GRADIENT;R: REAL) RES: GRADIENT;
     VAR I: DIMTYPE;M: REAL;U: GRADIENT;
     BEGIN U.F:=G.F**R;IF R = 1 THEN U.DF:=G.DF ELSE
           IF R = 0 THEN FOR I:=1 TO DIM DO U.DF[I]:=0 ELSE
           IF (G.F=0) AND (R > 1) THEN FOR I:=1 TO DIM DO U.DF[I]:=0 ELSE
           BEGIN M:=R*U.F/G.F;FOR I:=1 TO DIM DO IF G.DF[I] = 0 THEN
                 U.DF[I]:=0 ELSE U.DF[I]:=M*G.DF[I]
           END;
           RES:=U
     END;

OPERATOR ** (R: REAL;G: GRADIENT) RES: GRADIENT;
     VAR I: DIMTYPE;M: REAL;U: GRADIENT;
     BEGIN U.F:=R**G.F;IF (R=0) AND (G.F > 0) THEN FOR I:=1 TO DIM DO
           U.DF[I]:=0 ELSE
        BEGIN M:=U.F*LN(R);FOR I:=1 TO DIM DO IF G.DF[I] = 0 THEN
              U.DF[I]:=0 ELSE U.DF[I]:=M*G.DF[I]
        END;
        RES:=U
     END;

OPERATOR ** (K: INTEGER;G: GRADIENT) RES: GRADIENT;
     VAR R: REAL;U: GRADIENT;
     BEGIN R:=K;U:=R**G;
           RES:=U
     END;
```

```
OPERATOR ** (GA: GRADIENT;GB: GRADIENT) RES: GRADIENT;
    VAR I: DIMTYPE;R: REAL;U,V: GRADIENT;
    BEGIN R:=GA.F;U:=R**GB;R:=GB.F;V:=GA**R;
          FOR I:=1 TO DIM DO U.DF[I]:=U.DF[I]+V.DF[I];
          RES:=U
    END;
```

APPENDIX B. SOURCE CODE FOR STANDARD GRADIENT FUNCTIONS

(File GRAD_FUN.S)

B.1. Absolute Value

```
FUNCTION GABS(G: GRADIENT): GRADIENT;
    VAR I: DIMTYPE;M: REAL;U: GRADIENT;
    BEGIN U.F:=ABS(G.F);M:=G.F/U.F;FOR I:=1 TO DIM DO IF G.DF[I] = 0 THEN
        U.DF[I]:=0 ELSE U.DF[I]:=M*G.DF[I];
        GABS:=U
    END;
```

B.2. Square Root

```
FUNCTION GSQRT(G: GRADIENT): GRADIENT;
    VAR I: DIMTYPE;M: REAL;U: GRADIENT;
    BEGIN U.F:=SQRT(G.F);M:=2*U.F;FOR I:=1 TO DIM DO IF G.DF[I] = 0
        THEN U.DF[I]:=0 ELSE U.DF[I]:=G.DF[I]/M;
        GSQRT:=U
    END;
```

B.3. Exponential (Base e)

```
FUNCTION GEXP(G: GRADIENT): GRADIENT;
    VAR I: DIMTYPE;U: GRADIENT;
    BEGIN U.F:=EXP(G.F);FOR I:=1 TO DIM DO IF G.DF[I] = 0 THEN U.DF[I]:=0
        ELSE U.DF[I]:=U.F*G.DF[I];
        GEXP:=U
    END;
```

B.4. Natural Logarithm

```
FUNCTION GLN(G: GRADIENT): GRADIENT;
    VAR I: DIMTYPE;U: GRADIENT;
    BEGIN U.F:=LN(G.F);FOR I:=1 TO DIM DO IF G.DF[I] = 0 THEN U.DF[I]:=0
        ELSE U.DF[I]:=G.DF[I]/G.F;
        GLN:=U
    END;
```

B.5. Arctangent

```
FUNCTION GARCTAN(G: GRADIENT): GRADIENT;
    VAR I: DIMTYPE;M: REAL;U: GRADIENT;
    BEGIN U.F:=ARCTAN(G.F);M:=1+G.F*G.F;FOR I:=1 TO DIM DO IF G.DF[I] = 0
        THEN U.DF[I]:=0 ELSE U.DF[I]:=G.DF[I]/M;
        GARCTAN:=U
    END;
```

B.6. Sine

```
FUNCTION GSIN(G: GRADIENT): GRADIENT;
    VAR I: DIMTYPE;M: REAL;U: GRADIENT;
    BEGIN U.F:=SIN(G.F);M:=COS(G.F);FOR I:=1 TO DIM DO IF G.DF[I] = 0
        THEN U.DF[I]:=0 ELSE U.DF[I]:=M*G.DF[I];
        GSIN:=U
    END;
```

B.7. Cosine

```
FUNCTION GCOS(G: GRADIENT): GRADIENT;
    VAR I: DIMTYPE;M: REAL;U: GRADIENT;
    BEGIN U.F:=COS(G.F);M:=-SIN(G.F);FOR I:=1 TO DIM DO IF G.DF[I] = 0
        THEN U.DF[I]:=0 ELSE U.DF[I]:=M*G.DF[I]
    END;
```

APPENDIX C. OUTPUT OF THE GRADIENT PROGRAM FOR THE SOLUTION OF SYSTEM (9.10) BY NEWTON'S METHOD

```
INITIAL VALUES
      X = 1.00000000000E+00
      Y = 1.00000000000E+00
      Z = 1.00000000000E+00
FUNCTION VALUES
      F(X,Y,Z) = 1.70000000000E+01
      G(X,Y,Z) = 0.00000000000E+00
      H(X,Y,Z) = 0.00000000000E+00
CORRECTION VECTOR
      DX = -7.08333333334E-02
      DY = -2.12500000000E-01
      DZ =  2.83333333334E-01

VALUES FROM ITERATION NUMBER 1
      X = 9.29166666667E-01
      Y = 7.87500000000E-01
      Z = 1.28333333333E+00
FUNCTION VALUES
      F(X,Y,Z) = 4.79191713930E+00
      G(X,Y,Z) = 1.30451388890E-01
      H(X,Y,Z) = 1.46966869220E-02
CORRECTION VECTOR
      DX = -4.20921371897E-02
      DY = -9.43241406975E-02
      DZ =  3.75313068779E-02

VALUES FROM ITERATION NUMBER 2
      X = 8.87074529477E-01
      Y = 6.93175859303E-01
      Z = 1.32086464021E+00

ITERATION NUMBER 2 (CONTINUED)
FUNCTION VALUES
      F(X,Y,Z) = 6.45309522200E-01
      G(X,Y,Z) = 1.20773905300E-02
      H(X,Y,Z) = 4.86417092500E-03
CORRECTION VECTOR
      DX = -8.83013113461E-03
      DY = -1.59811519852E-02
      DZ =  9.74516049898E-03

VALUES FROM ITERATION NUMBER 3
      X = 8.78244398342E-01
      Y = 6.77194707318E-01
      Z = 1.33060980071E+00
FUNCTION VALUES
      F(X,Y,Z) = 1.84509451000E-02
      G(X,Y,Z) = 4.28336590000E-04
      H(X,Y,Z) = 2.06810341000E-04
CORRECTION VECTOR
      DX = -2.78405682738E-04
      DY = -4.37403612547E-04
      DZ =  2.45411801068E-04

VALUES FROM ITERATION NUMBER 4
      X = 8.77965992659E-01
      Y = 6.76757303705E-01
      Z = 1.33085521251E+00
FUNCTION VALUES
      F(X,Y,Z) = 1.47972000000E-05
      G(X,Y,Z) = 3.29050000000E-07
      H(X,Y,Z) = 2.04196000000E-07
```

ITERATION NUMBER 4 (CONTINUED)

CORRECTION VECTOR

DX = -2.32383617048E-07

DY = -3.33184805859E-07

DZ = 1.99108934899E-07

VALUES FROM ITERATION NUMBER 5

X = 8.77965760275E-01

Y = 6.76756970520E-01

Z = 1.33085541162E+00

FUNCTION VALUES

F(X,Y,Z) = 1.00000000000E-10

G(X,Y,Z) = 0.00000000000E+00

H(X,Y,Z) = -1.00000000000E-12

CORRECTION VECTOR

DX = -1.18197463333E-12

DY = -3.73328280534E-12

DZ = 2.67817103788E-12

VALUES FROM ITERATION NUMBER 6

X = 8.77965760274E-01

Y = 6.76756970516E-01

Z = 1.33085541162E+00

FUNCTION VALUES

F(X,Y,Z) = 0.00000000000E+00

G(X,Y,Z) = 0.00000000000E+00

H(X,Y,Z) = 2.00000000000E-12

CORRECTION VECTOR

DX = -4.18558019594E-13

DY = 1.03209645475E-12

DZ = -2.48711360538E-13

VALUES FROM ITERATION NUMBER 7

X = 8.77965760274E-01

Y = 6.76756970517E-01

Z = 1.33085541162E+00

FUNCTION VALUES

F(X,Y,Z) = 0.00000000000E+00

G(X,Y,Z) = 0.00000000000E+00

H(X,Y,Z) = 1.00000000000E-12

CORRECTION VECTOR

DX = -2.09279009797E-13

DY = 5.16048227375E-13

DZ = -1.24355680269E-13

VALUES FROM ITERATION NUMBER 8

X = 8.77965760274E-01

Y = 6.76756970518E-01

Z = 1.33085541162E+00

FUNCTION VALUES

F(X,Y,Z) = 0.00000000000E+00

G(X,Y,Z) = 0.00000000000E+00

H(X,Y,Z) = 0.00000000000E+00

CORRECTION VECTOR

DX = 0.00000000000E+00

DY = 0.00000000000E+00

DZ = 0.00000000000E+00

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 2400	2. GOVT ACCESSION NO. AD-A120 959	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) DIFFERENTIATION IN PASCAL-SC: TYPE GRADIENT		5. TYPE OF REPORT & PERIOD COVERED Summary Report - no specific reporting period 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) L. B. Rall		8. CONTRACT OR GRANT NUMBER(s) DAAG29-80-C-0041
9. PERFORMING ORGANIZATION NAME AND ADDRESS Mathematics Research Center, University of Wisconsin 610 Walnut Street Madison, Wisconsin 53706		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Work Unit Number 3 - Numerical Analysis and Computer Science
11. CONTROLLING OFFICE NAME AND ADDRESS U. S. Army Research Office P.O. Box 12211 Research Triangle Park, North Carolina 27709		12. REPORT DATE July 1982 13. NUMBER OF PAGES 33
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Scientific computation, Automatic differentiation, Gradients, Jacobians, Sensitivity analysis, Error analysis, Optimization, Nonlinear systems of equations, PASCAL-SC

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

In scientific computation, there is often need for the derivatives as well as the values of functions defined by computer programs. Here, it is shown how automatic differentiation can be carried out in a modern computer language which permits user-defined operators and data types. The specific language used is PASCAL-SC, and differentiation is implemented for variables of type GRADIENT, which consists of the value of a function of n real variables and its gradient vector of first partial derivatives with respect to the independent variables.

(continued)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ABSTRACT (continued)

Calculations of the results of operators or functions applied to GRADIENT variables are carried out according to the well-known rules for evaluation and differentiation of sums, differences, products, and so on. Since the differentiation is performed at compile time, the code produced is comparable in compactness and execution time to that obtained if numerical approximations are used for derivatives, and the theoretical and practical problems associated with numerical differentiation are avoided. PASCAL-SC source code is given for the necessary operators and standard functions, and it is shown how to prepare code for arbitrary differentiable functions to add to the library if desired. The effectiveness of the use of type GRADIENT is shown by an example of the solution of a system of nonlinear equations by Newton's method.